CS 458 A1-MILESTONE

Chia Ching Chuen |WATID: CCCHIA | STUDENT NUMBER: 20755359

Abstract

Explanation of exploit 1 and exploit 2

Sploit1.c

This exploit program will overflow the option -s of the program pwgen. As the program code for the parameter -s, accepts a user's input for the insertion of the string, it seems like a good place to start trying to exploit the program. Firstly, after looking through the code of the pwgen, in the code, I find that there exists a line of code :

```
args = parse_args(argc argv);
```

Inside of parse_args function, specifically, the line:

strcpy(args.salt, optarg);

as strcpy doesn't check whether the string variable is big enough for the string to be copied over, we are able to use this for a buffer overflow attack to change the return address after the function ends.

For now, the location of the return address, the amount of space to overwrite with NOPs and the shellcode before inserting the address of the location to jump to and the address to jump to is needed. first, I have to know where args.salt is located at specifically, using gdb, we are able to find the location at 0xffbfd7df by using the "print &args.salt" command.

As we checked, the distance from salt to the return address is roughly 557, we construct the buffer with a size of 557 bytes.

We first constructing the buffer with NOP, the shellcode and the address of args.salt repeatedly, to jump to in order. This will be our payload and make the RA hopefully jump to the NOP or the shellcode.

By now, after compiling the program, and running it, a segmentation fault will be shown where my return address is a jumbled up of 0xffbfd7df address. After wrapping the address around and compiling it, the vulnerability is exploited, and a root shell is spawned.

To fix this vulnerability, simply use a check instruction before the strcpy instruction so that the string will stop copying whenever the buffer is full or simply print an error message when the string provided is larger than the salt buffer size.

For example:

```
If (strlen(optarg) < 555)
        strcpy(args.salt, optarg)
else{
        printf("provided input too big");
        exit(0);
}
```

Sploit2.c

In print_usage function that takes in the argument 0, we see that first, it creates a char buffer which can contain 656 characters. After that, it copies the help menu, and the argument into buffer. In this case, the sprintf function do not check the size of the argument and insert it into the %s in the help menu. This allows us to potentially overflow the buffer parameter and overwrite the return address of print_usage.

First we get our payload ready, which will be the same payload as sploit1. After that, we insert it into argument 0 instead of argument 2. Then to make that we enter print_usage, we will use the -h

parameter for argument 1. After finishing setting the parameters properly, running the program will grant us a root shell.

This can be solved like the exploit above, where we will first check if the string provided exceed the size of the buffer, if the size does not exceed it, proceed else, print an error message for the user and exit the program.